

Create a Servlet in Eclipse

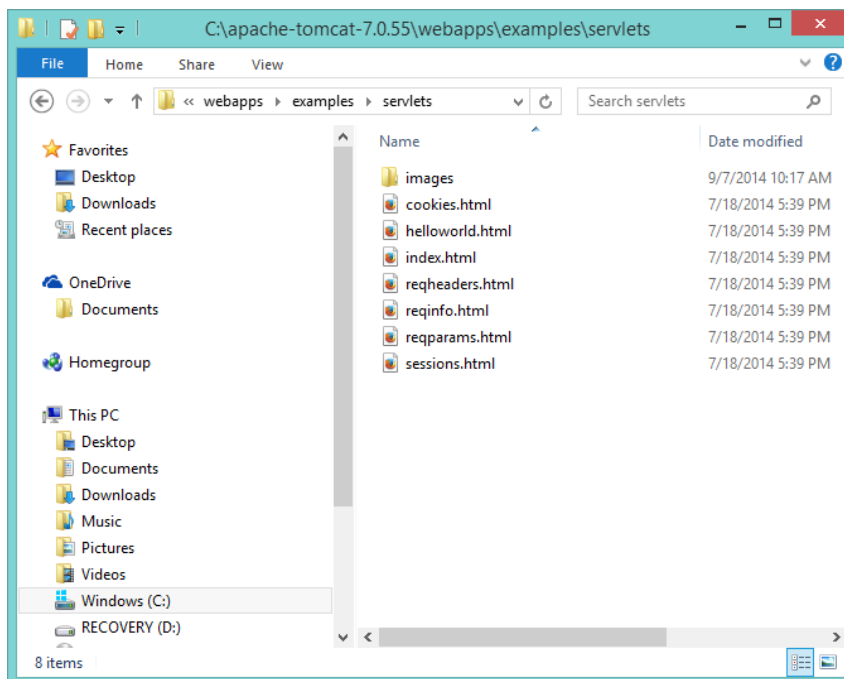
Tomcat has several examples of Servlets. In this document one of these example Servlets will be used to create a sample servlet in Eclipse. Note that the code that will be used is from the Tomcat Project and resides within the distribution for Tomcat. It is in the Tomcat webapps/examples folder.

The steps in this exercise will be to

- Create a servlet in Eclipse
- Run the servlet in Eclipse on Tomcat and view the results
- Export a WAR from Tomcat using our project
- Install the WAR on Tomcat
- Run Tomcat and view the Servlet results without Eclipse
- Explore the expanded WAR's contents on Tomcat
- Explore the Servlet in the Eclipse workspace

The point of the exercise is to see how a Servlet in a WAR is created by the development team using Eclipse, and to see the differences once a WAR is created and placed on Tomcat. Note that other IDEs such as Netbeans have a similar development, deployment cycle.

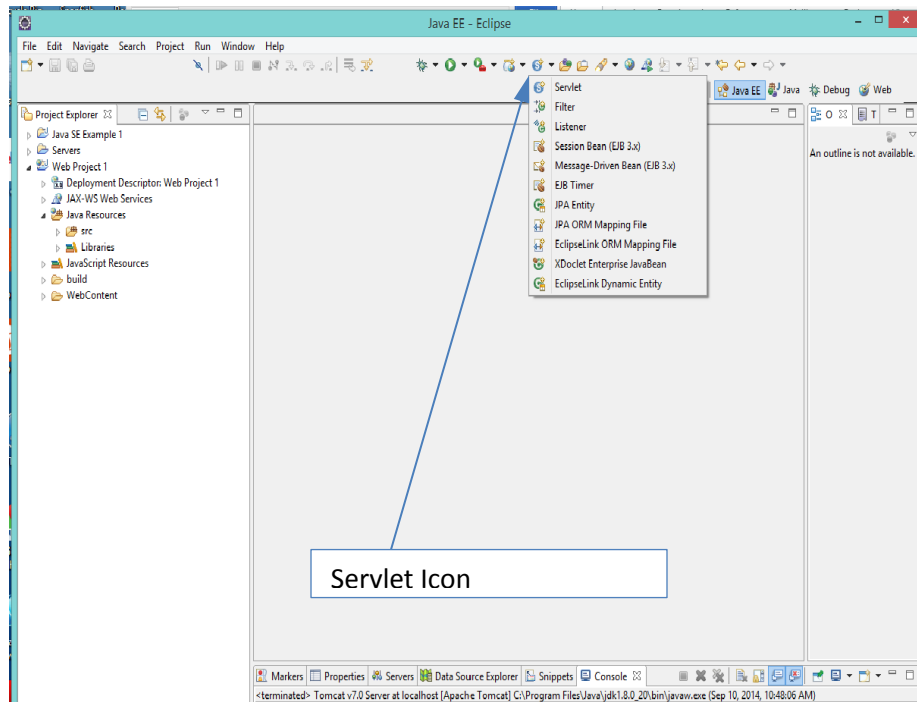
First go to the Tomcat Installation that exists on your computer. Tomcat was setup in a prior exercise. For example on this computer the webapps/examples/servlets folder is shown below.



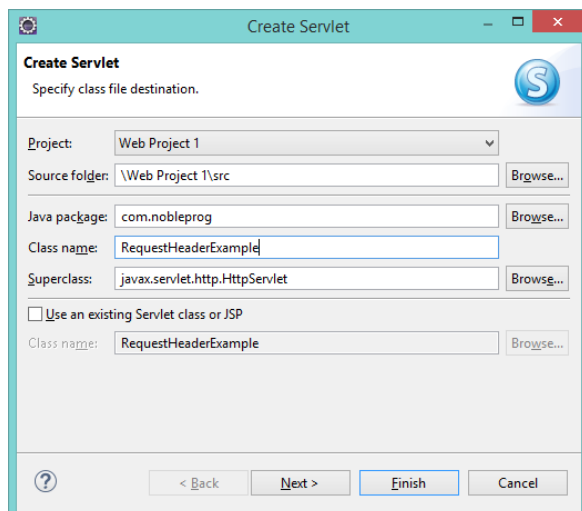
To set up a servlet in Eclipse the code for the reqheaders.html file will be used once we have created a servlet with a doGet method in Eclipse.

To create a servlet on Eclipse:

- Open Eclipse
- Get into the new web project created in a prior example
- Make sure that the EE perspective is being displayed in the Project Explorer (perspectives can be changed under menu item “Windows”)
- Click on the Servlet Icon as shown in the snapshot below



Clicking on the Servlet Icon or by opening the dropdown menu and choosing Servlet will bring up the Dialog:



Note that the source will be placed in the src folder. In this case the Servlet is to be placed in the package com.nobleprog, and the name will be RequestHeaderExample which is the name given in the Tomcat Example. Clicking on Next will give the next Dialog where the deployment descriptor information can be given. The Dialog is shown on the next page.

Below is given the Eclipse Dialog where the deployment Descriptor information for a Servlet can be enter.

The 'Create Servlet' dialog box is shown with the title 'Create Servlet'. The subtitle is 'Enter servlet deployment descriptor specific information.' The dialog contains the following fields and controls:

- Name:** A text field containing 'RequestHeaderExample'.
- Description:** An empty text field.
- Initialization parameters:** A table with columns 'Name', 'Value', and 'Description'. It is currently empty. To the right of the table are buttons: 'Add...', 'Edit...', and 'Remove'.
- URL mappings:** A text area containing '/RequestHeaderExample'. To the right are buttons: 'Add...', 'Edit...', and 'Remove'.
- Asynchronous Support:** A checkbox that is currently unchecked.
- Navigation buttons:** At the bottom are buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

The URL mapping can be changed here here is so desired. When a request comes to the Tomcat Servlet Container the requested URL is parsed. The last part of the URL used to determine which Servlet (or JSP page) is to be used to service the request.

Clicking on Next brings up the Dialog shown below.

The 'Create Servlet' dialog box is shown with the title 'Create Servlet'. The subtitle is 'Specify modifiers, interfaces to implement, and method stubs to generate.' The dialog contains the following fields and controls:

- Modifiers:** Checkboxes for 'public' (checked), 'abstract', and 'final'.
- Interfaces:** An empty text area. To the right are buttons: 'Add...' and 'Remove'.
- Which method stubs would you like to create?** A group of checkboxes:
 - ☒ Constructors from superclass
 - ☒ Inherited abstract methods
 - ☐ init
 - ☐ destroy
 - ☐ getServletConfig
 - ☐ getServletInfo
 - ☐ service
 - ☒ doGet
 - ☒ doPost
 - ☐ doPut
 - ☐ doDelete
 - ☐ doHead
 - ☐ doOptions
 - ☐ doTrace
- Navigation buttons:** At the bottom are buttons: '?', '< Back' (highlighted with a blue border), 'Next >', 'Finish', and 'Cancel'.

It is known from our example that only the doGet method is going to be used in our Servlet. Therefore we should uncheck the doPost method stub so that it is not generated.

The method doGet handles requests that come in with query strings after a question mark behind the URL. For example, the URL

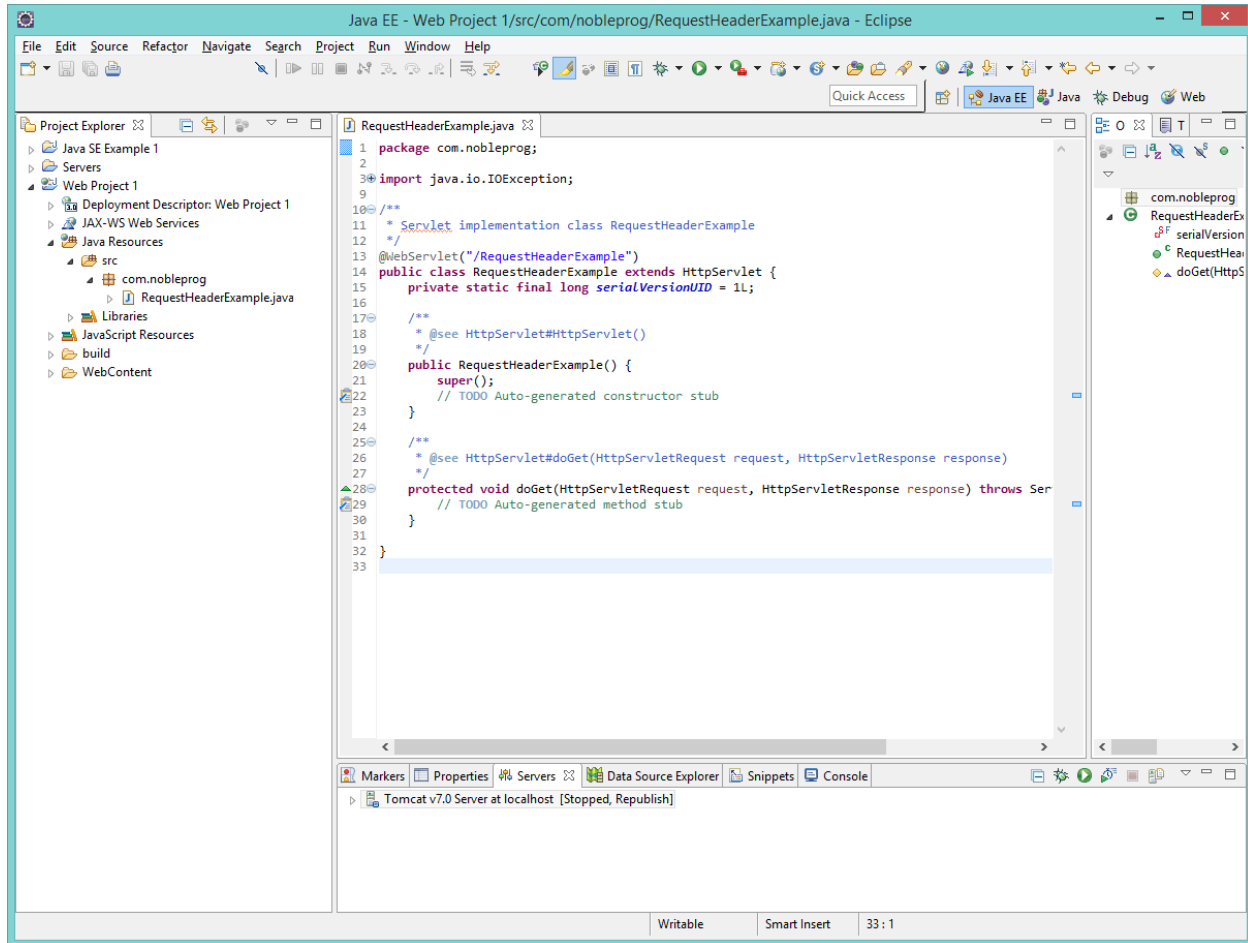
<http://something?parameter=5>

has a query string behind the URL

On the other hand a doPost method usually handles a request from a HTML form where the form method="POST".

A get request has a limit (2048 characters) on the amount of data that can be passed where a post does not. In this case a get will be used since in our test a URL without any data will be used. Therefore uncheck the doPost method. If doPost is left checked it will not affect the outcome since an empty method stub will just be created which can just be left alone since it will not be called.

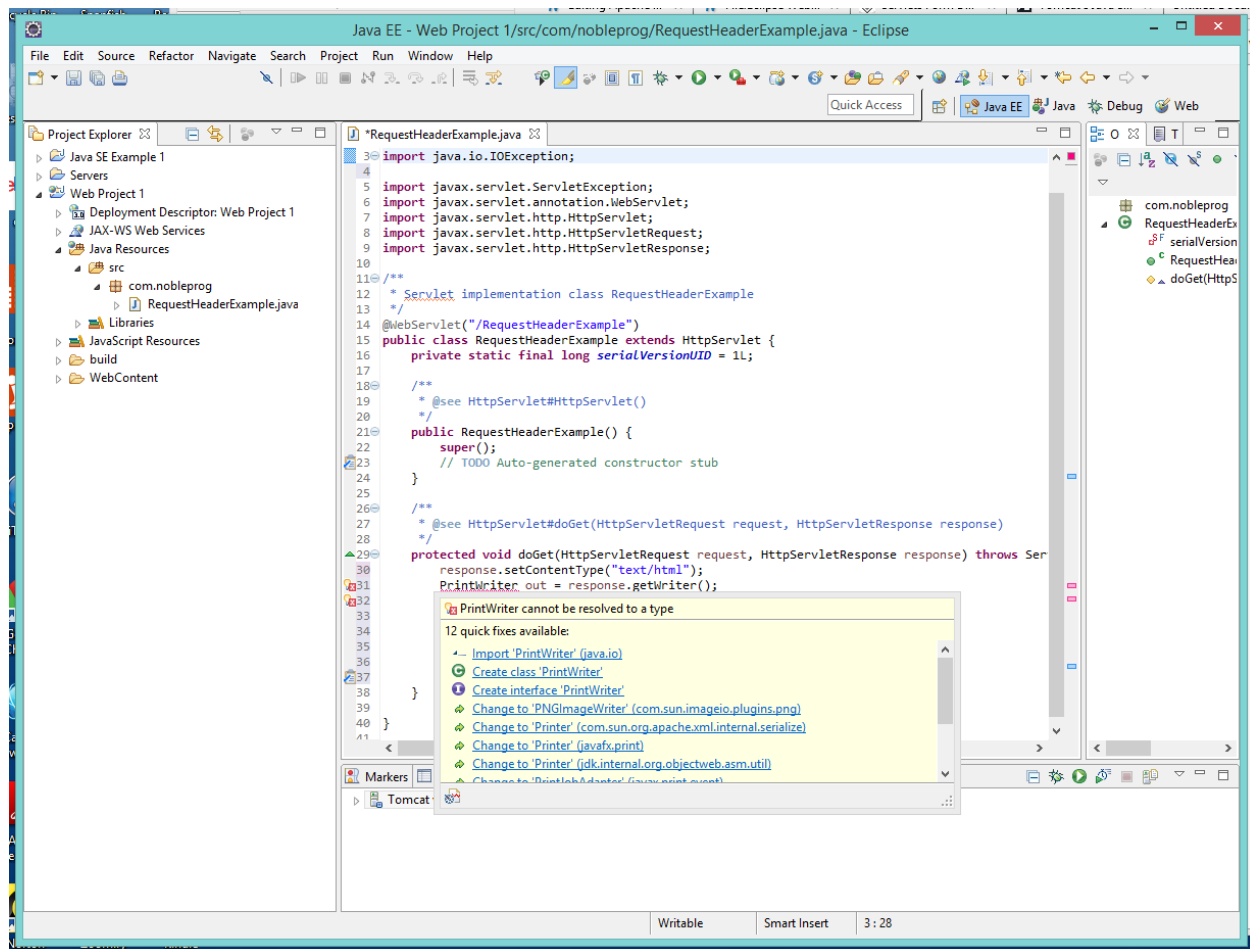
Clicking Finish creates the Servlet as shown below.



Now open the Tomcat Example Servlet called “reqheaders.html” and copy and paste the code inside the doGet method into our Eclipse example doGet method.

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
Enumeration e = request.getHeaderNames();
while (e.hasMoreElements()) {
    String name = (String)e.nextElement();
    String value = request.getHeader(name);
    out.println(name + " = " + value);
}
```

Note that you get two errors in the code since `PrintWriter` and `Enumeration` are not known to the compiler. If you roll over `PrintWriter` with the cursor you will get a popup which will tell how to correct the error as shown below:



In the popup click on “Import PrintWriter (java.io)” which will put an Import statement in the file correcting the PrintWriter error. Do a similar operation for Enumeration placing an import statement in the file for it. This will correct the error in the file allowing it to compile.

The file is now:

```
package com.nobleprog;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

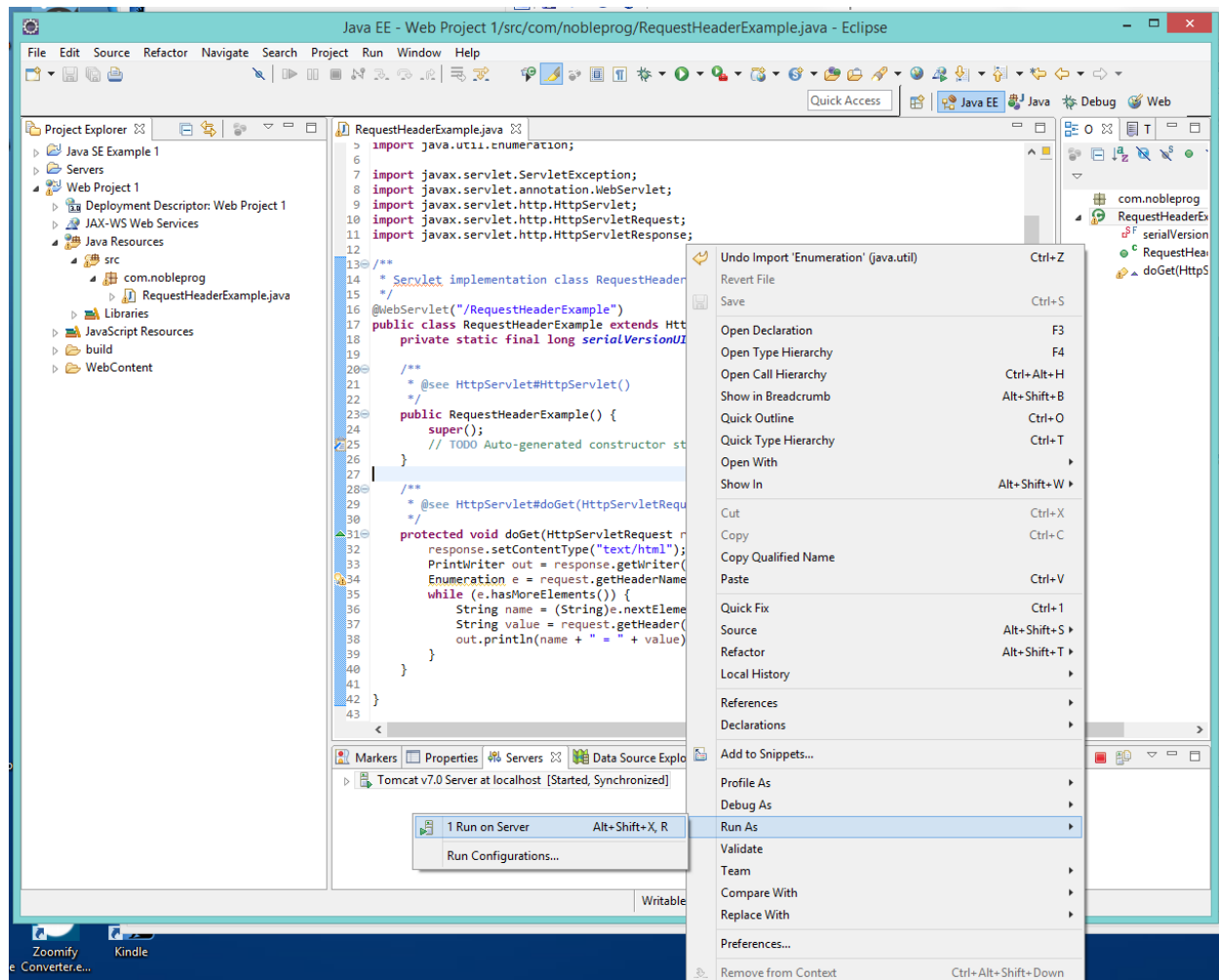
/**
 * Servlet implementation class RequestHeaderExample
 */
@WebServlet("/RequestHeaderExample")
public class RequestHeaderExample extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public RequestHeaderExample() {
        super();
        // TODO Auto-generated constructor stub
    }

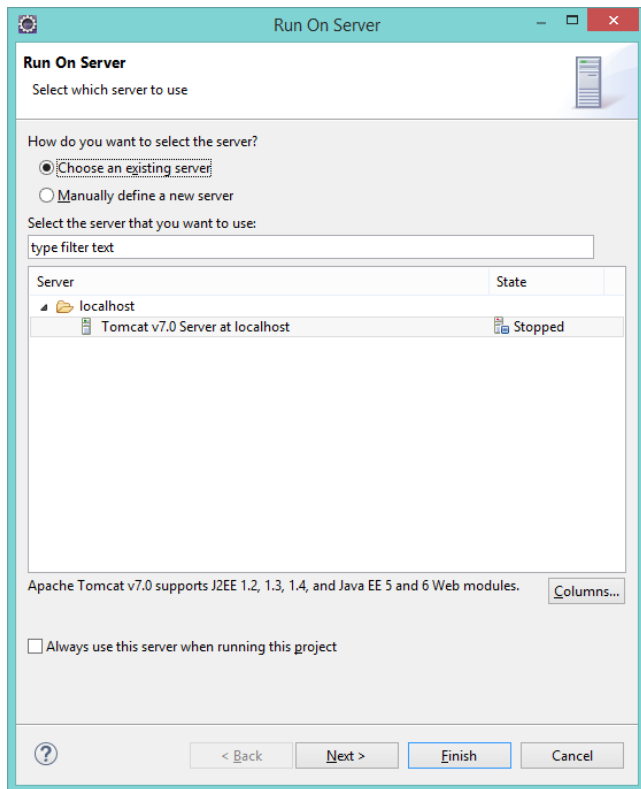
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
    response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Enumeration e = request.getHeaderNames();
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = request.getHeader(name);
            out.println(name + " = " + value);
        }
    }
}
```

Now save the file using the Save icon or by right clicking and choosing Save.

Right Clicking on the window in Eclipse in which RequestHeaderExample.java is displayed brings up a popup menu in which RunAs/Run on Server can be chosen.

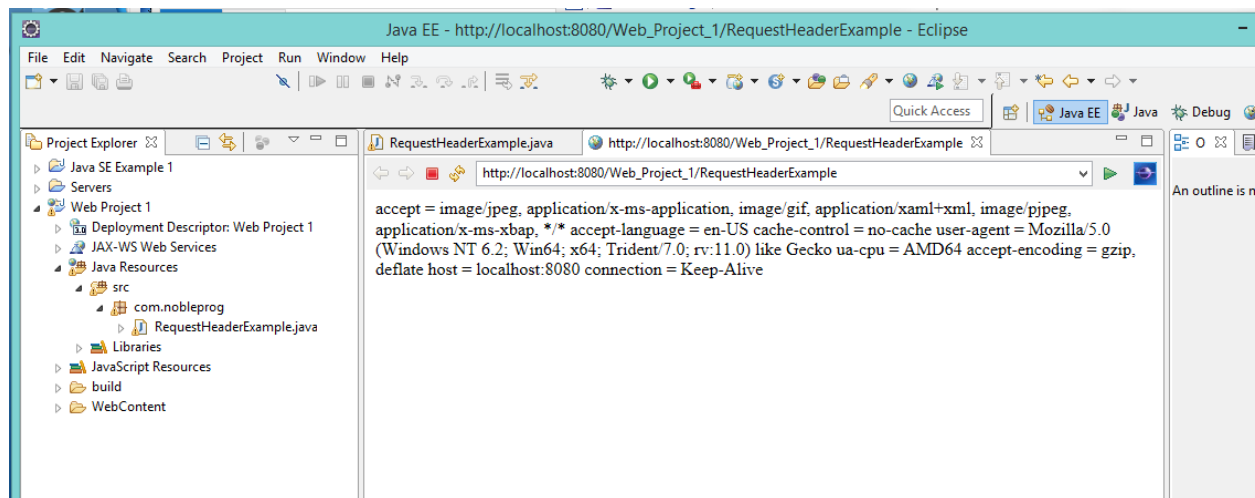


Clicking on “Run on Server” brings up the following Dialog:



Click Finish to Run Tomcat and request the Servlet.

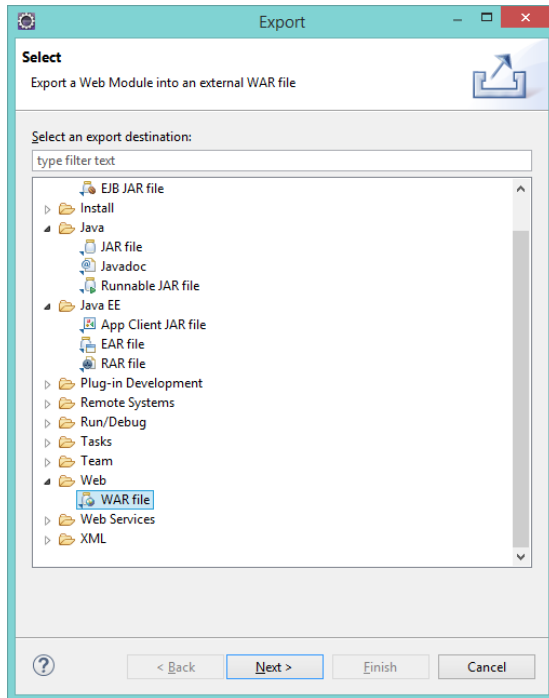
The result is shown in the image below. You can also take the URL given in the image and put it into a browser such as Firefox or Internet Explorer and get the same results.



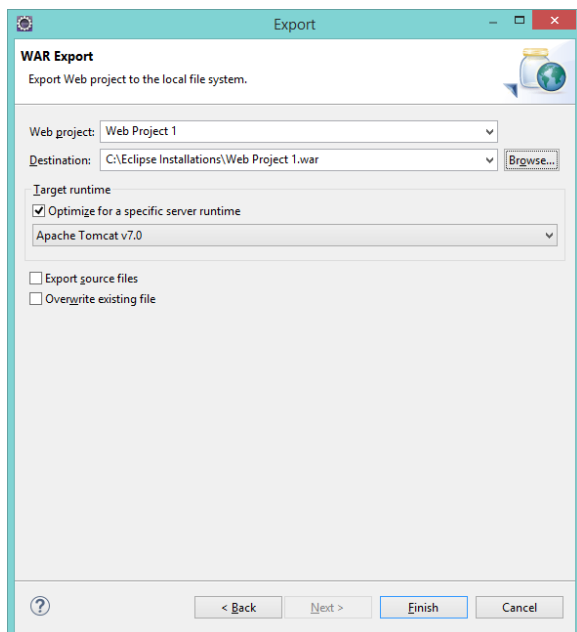
The results show the headers of the HTTP request.

Exporting a WAR from Eclipse

To make a WAR file from the project click on File/export which gives the following Dialog



Choose Web/WAR file from the Dialog and Click Next. This gives the Dialog below:



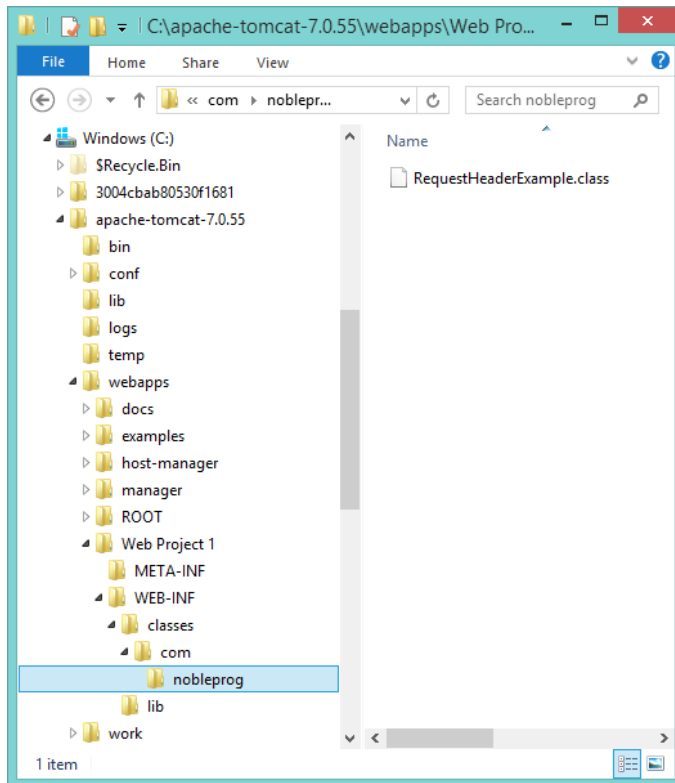
Browse and find a location for the WAR which can be found later. Then click Finish which will save the WAR to the chosen Destination.

The WAR can be installed in Tomcat by placing it in the webapps folder for the installation. When Tomcat is started the WAR will be unpacked and placed as a folder in the webapps folder. Going to the

URL in a Browser as done before will now pick up the servlet in the webapps folder and run it to service the request.

Exploring the Tomcat WAR folders vs the Eclipse Folders

When the WAR is unpacked by Tomcat the folder/directory structure looks like the image below:



The class file for the servlet that was created is now located in the WEB-INF/classes folder under the folders representing its package which was nobleprog.com. There is a Manifest file in META-INF but it is just a stub that gives only the version of the manifest.

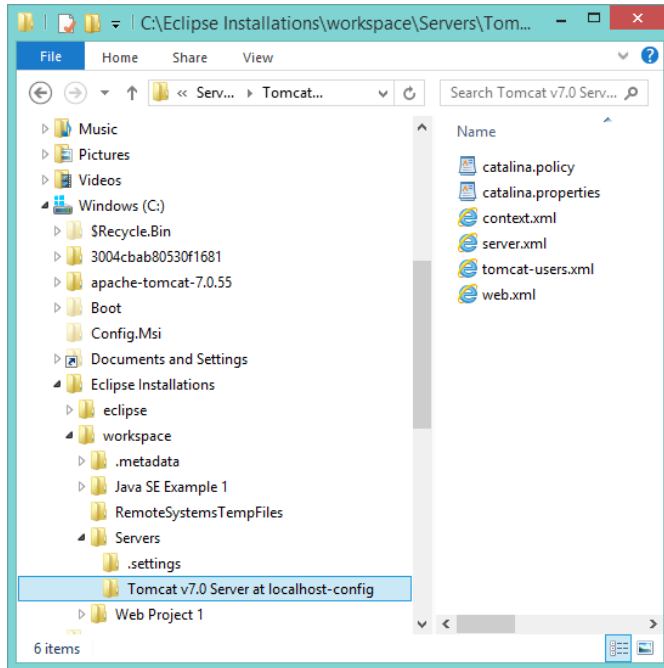
In Tomcat 7 annotations in the code can be used to define Servlets rather than using a web.xml file. This has occurred in this example. The code from Apache Tomcat includes the following lines:

```
@WebServlet("/RequestHeaderExample")
public class RequestHeaderExample extends HttpServlet {
```

This means that for this example a web.xml file does not need to exist in Tomcat

Note that if there are servlets that do not have WebServlet annotations in the system then web.xml will still exist to define those servlets to Tomcat.

If we look at the Eclipse definitions that allow the web project to run they are located in:
C:\Eclipse Installations\workspace\Servers\Tomcat v7.0 Server at localhost-config as shown below/



In particular the server.xml file defines the following

```
<Host name="localhost" unpackWARs="true" autoDeploy="true" appBase="webapps">
  <!-- SingleSignOn valve, share authentication between web applications Documentation at:
   /docs/config/valve.html -->
  <!-- <Valve className="org.apache.catalina.authenticator.SingleSignOn" /> -->
  <!-- Access log processes all example. Documentation at: /docs/config/valve.html Note: The pattern
   used is equivalent to using pattern="common" -->
  <Valve className="org.apache.catalina.valves.AccessLogValve" suffix=".txt"
    prefix="localhost_access_log." pattern="%h %l %u %t \"%r\" %s %b" directory="logs"/>
  <Context source="org.eclipse.jst.jee.server:Web Project 1" reloadable="true"
    path="/Web_Project_1" docBase="Web Project 1"/>
</Host>
```

The Context element defines a web application that is run on a particular virtual host from either a WAR file or a folder on that host. Looking at the Context element in the Host called localhost the elements below are defined in the context.

The Context path

This definition gives the host localhost a context source path of /Web_Project_1. This defines the part of the URL that is after the host name and port. For example the URL:

http://localhost:8080/Web_Project_1/RequestHeaderExample

has a path of Web_Project_1 and a WebServlet name of /RequestHeaderExample which aligns with the annotation in the code.

The Context docBase

The docBase aligns the path from the URL with a WAR file or folder on the host. The docBase names the WAR file or folder.

The Context source

The source attribute is not a Tomcat attribute and is used by Eclipse (The Eclipse Web Tools Platform) to match the running application to a project in the Eclipse workspace. Tomcat actually gives a warning stating that it cannot match the property to a property that it knows. However, since it is just a warning Tomcat runs correctly.

Eclipse passes the source to Tomcat when it starts it. The .classpath under the project gives the name of the folder where the servlet class files are located and this allows Tomcat to run the correct servlet. Tomcat determines the servlet to run using the name of the project, and the name given in the code annotation (since annotations can be used in Tomcat 7).

Conclusion

The servlet runs correctly on Tomcat whether it is run from the development environment in Eclipse or from a WAR file placed on Tomcat in its webapps folder.

Running the servlet from Eclipse is much more complex in the behind the covers settings and manipulations. However, essentially the two methodologies work the same.